Low Latency Parallel Schedulers for Photonic Integrated Optical Switch Architectures in Data Centre Networks

Paris Andreades and Philip M. Watts

Optical Networks Group, Department of Electronic and Electrical Engineering, University College London, London, WC1E 7JE, paris.andreades.09@ucl.ac.uk

Abstract Using speculative transmission combined with a novel parallel scheduler design for practical photonic integrated switches based on the Clos architecture, we demonstrate a minimum latency of 47.2 ns in a rack scale 32x32 optically switched system.

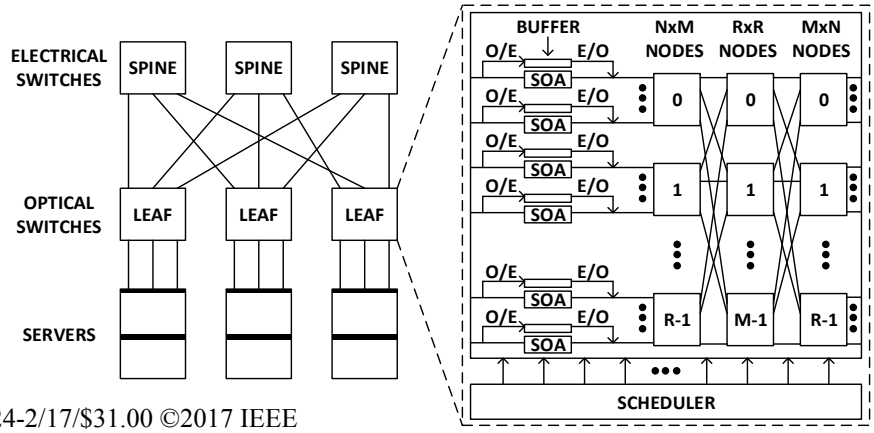
Introduction

Global data centre IP traffic has been growing at a rate of 27% every year since 2015 and will continue to do so reaching a total of 15.3 zettabytes by the end of 2020[1], with low latency applications such as high frequency trading and telemedicine assuming greater importance. This requirement is reflected in the state-of-the art data centre networks which are shifting from the traditional multi-tiered model to the leaf-spine topology in order to flatten the architecture and offer low and bounded network latency. However, the fastest electronic switches that can be used in this topology have a latency on the order of 200 ns[2]. Also, their bandwidth is limited by the number of high-speed signal pins on the chip[3].

Optical circuit switching[4] is now deployed in data centres but has millisecond reconfiguration times. Networks based on nanosecond optical switches with speculative transmission to reduce latency are an attractive solution[5,6]. Recently we demonstrated 75 ns end-to-end latency using speculative transmission in a rack scale optical network control plane demonstration with a central scheduler[7]. However, the demonstration assumed a crossbar optical switch. $K \times K$ crossbar switches are not suitable for photonic integration because they require K^2 components[8] and a *K*-way optical power split per data path which leads to poor optical signal-to-noise ratio (OSNR) performance. A more feasible alternative is to arrange small switching nodes in multi-stage topologies, such as Clos and butterfly networks. However, unlike a crossbar switch, scheduling a Clos-network switch requires a route calculation function. The traditional software *looping* routing algorithm[9] rearranges the current routing matrix to add new entries, iterating to resolve any subsequent conflicts that arise. Such an algorithm is unsuitable for fast speculative switching. In this paper we present the design of a novel low latency parallel scheduler for Clos-network switches and we evaluate its performance compared with a crossbar scheduler in terms of minimum latency and maximum throughput.

System Concept

Our previously published low latency speculative control plane[10] is shown in Fig. 1. At the network edge, all new packets are first stored in FIFO queues at the servers. Any server network interface may issue an output-port and path request to the scheduler, on dedicated control wires, for the head-of-line (HOL) packet in its queue. The packet is transmitted speculatively in parallel WDM form using WDM transceivers after the guard time has elapsed so that it arrives at the switch as soon as it is configured, in the case of successful allocation. Each input port of the switch has a dedicated WDM receiver and a FIFO buffer to avoid packet dropping. If a packet fails allocation, due to output port or path contention, the scheduler asserts the FIFO write-enable and buffers the packet for transmission in a subsequent cycle. The buffer controller issues a request to the scheduler for the HOL packet and releases it only when the scheduler asserts the corresponding read-enable line. The scheduler



978-1-5386-5624-2/17/$31.00 ©2017 IEEE

Fig. 1: Data Centre Network and Proposed Optical Switch (m,n,r) Clos Architecture

gives priority to buffer requests over new ones form the server interfaces. This maintains the packet ordering and reduces average latency.

The switch architecture has low latency, compared with electronic switches, due to (1) low serialisation latency by high bandwidth WDM ports, (2) speculative transmission and (3) fast parallel hardware scheduling. The latter is the subject of this paper.

Scheduler Design

In this work, we divide the Clos scheduler circuit into 3 pipeline stages: (1) *Output-port Allocation*, (2) Clos *Routing* and (3) *Configuration Update*, as shown in Fig. 2. In this way paths are granted in 3 clock cycles.

In the first pipeline stage, two output-port allocators are implemented; one for the server requests and one for the buffer requests. Each $K \times K$ allocator is built as a set of *K*-bit round-robin arbiters. Performing allocation separately for the servers and the buffers enables giving priority to buffer grants without using feedback logic that increases the critical path. This process is identical for both the crossbar and Clos schedulers. Implementing this global output port allocation process for the Clos switch ensures global fairness across all ports. In the case of the Clos, the request matrices for routing the input-stage (IS) and middle-stage (MS) are also generated in this first pipeline stage: an $M \times N$ matrix for every IS node and an $R \times R$ matrix for every MS node. This is performed separately for the servers and the buffers and then priority logic is used to filter out any server requests that contend with buffer requests. The key decision in Clos routing is the choice of middle stage switch. In the software looping algorithm, paths are iteratively rearranged to achieve near optimum matching. In order to achieve parallel routing in a single clock cycle, our scheduler chooses the middle switch for each request randomly. The drawback of this design choice is that the saturation throughput of the network is dependent on the number of middle-stage switches, as can be observed from the results below. Random middle stage switch selection trades off saturation throughput for minimum latency at low loads. The filtered IS and MS matrices are registered and forwarded to the next pipeline stage.

The routing function is performed as a distributed parallel path allocation; a dedicated path allocator is used for every switch node at the input-stage and middle-stage of the Clos network fabric. Hence, *R* allocators process the IS request matrices and *M* allocators process the MS requests matrices to generate path grants. All the

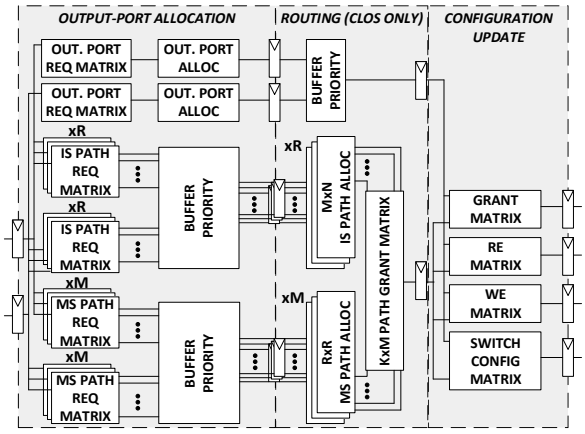


Fig. 2: Scheduler Design for (m,n,r) Clos Fabrics

allocators operate in parallel and hence the critical path in the routing pipeline stage reduces to the time needed to propagate through one of the arbiters for the input or middle stage, whichever ones are larger depending on the Clos configuration chosen. The path grants from both stages are then used to generate a global $K \times M$ path grant matrix which indicates which path has been granted per switch input port through the entire fabric.

In the third and final stage of the pipeline, the next switch configuration is computed using both the output-port grant matrix and the global path grant matrix. Any server/buffer requests winning both output-port and path allocation will lead to grants/read-enable signals and new switch configuration whereas any failing server requests will lead to FIFO write-enable signals.

Depending on the Clos network configuration, the critical path in the design is either in the 1st or 3rd pipeline stage. Carry-look-ahead logic is implemented to shorten the critical path in the first stage.

Results and Discussion

The scheduler design for a 32x32 switch was implemented on the Virtex-7 XC7VX690T for both crossbar and different Clos fabrics. For Clos, we chose (m=4,n=4,r=8) which has been shown to be highly attractive for fabrication[9] as well as 3 configurations with a higher number of middle-stage switches: (8,4,8), (8,2,16) and (16,2,16).

The implementation results, are shown in Figs. 3(a) and 3(b). The lookup-table (LUT) utilization is a measure of the scheduler circuit size on the FPGA. The minimum clock period, determining the latency, achievable for both (4,4,8) and (8,4,8) Clos is 5.4 ns, compared to 5 ns for the crossbar. The small increase is due to the path request generator logic which is not required in the crossbar scheduler, an effect which is exacerbated in the larger Clos switches. As there is a parallel allocator for each switch

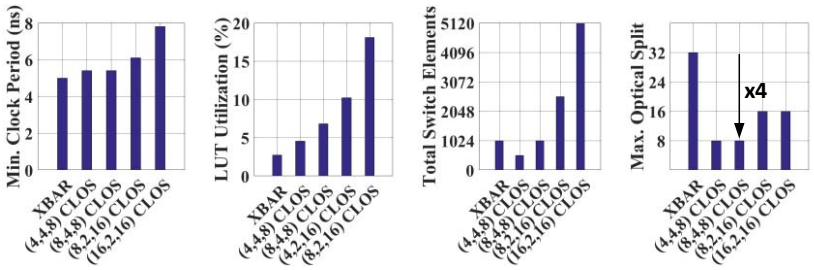


Fig. 3: Complexity comparison between crossbar and various Clos configurations (a) Minimum clock period of the scheduler (b) Lookup table utilization of the scheduler in an FPGA implementation (c) Number of switch elements, e.g. SOAs (d) Maximum optical split per data path

element, the FPGA LUT resources also increase substantially for these larger Clos switches. Figures 3(c) and 3(d) show the number of SOA switching elements required to build the switches and the maximum optical split that occurs per data path, as metrics of the switch fabrication feasibility. The crossbar has 1024 SOA elements and a very large 32-way split leading to poor OSNR performance making it impractical for photonic integration. On the other hand, the (4,4,8) Clos uses 512 elements but only a maximum 8-way split at the middle-stage, 4 times less compared to the crossbar, and thus it is very attractive for fabrication[8]. The (8,4,8) Clos needs 1024 SOAs but still the same split. The (8,2,16) and (16,2,16) Clos configurations are not feasible mainly because they require too many SOA components. However, we still study them to evaluate the routing algorithm performance.

Figure 4 shows the average end-to-end latency for random traffic, following the previously published method[7,10], for a rack-scale optical switching system for 64B packets serialized at 200 Gb/s using 8-wavelength WDM, based on the clock period values from Fig. 3(a). The crossbar scheduler functionality is identical to the Clos but without the routing stage. Therefore, the latency results are a measure of our parallel routing algorithm performance with the crossbar curve as the baseline. At low loads, the (4,4,8) and (8,4,8) Clos achieve 47.2 ns minimum latency, 16% higher than the crossbar scheduler (40.6 ns), due to the extra routing stage and the slightly longer clock period. The saturation throughput of the (4,4,8) Clos is only 32% but as the number of middle-stage switches is increased, in the other Clos configurations, we approach the crossbar performance as this decreases contention probability in our random path assignment scheme. The (8,4,8) Clos gives a good balance between viable switch architecture and latency performance, saturating at 50% throughput against 66% for the crossbar case. Both the crossbar and Clos scheduler designs give better latency performance than the previously reported 75 ns figure[7] because of faster scheduler design, synchronous control plane communication and lower packet serialization latency.

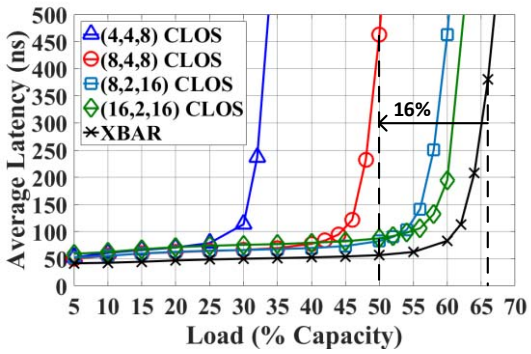


Fig. 4: Latency Performance for 32x32 Switch Fabrics

Conclusions

Multi-stage optical switch fabrics such as Clos are suitable for photonic integration due to reduced optical component counts and splitting losses. However, scheduling such fabrics requires a route calculation function incurring long latencies using the traditional looping algorithm. We presented a fast highly-parallel scheduler design for Clos network fabrics which trades off switch complexity for ultra-low latency. For a 32x32 rack-scale optically switched system our scheduler enables a 47.2 ns minimum end-to-end latency and a saturation throughput of 50% incurring a small routing penalty of 16% in both minimum latency and saturation throughput compared to an equivalent crossbar scheduler.

References

[1] *Cisco Global Cloud Index*, (2016).

[2] *Cisco Nexus 3548 Switch Performance Validation* (2012).

[3] N. Zilberman et. al., Proc. IEEE, Vol. **103**, p. 1102 (2015).

[4] N. Farrington et al., SIGCOMM, (2010).

[5] Luijten et. al., Proc. ACM/IEEE Supercomputing, (2005).

[6] A. Shacham et al., *IEEE Micro*, Vol. **27**, p.6 (2007).

[7] P. Andreades et al., OFC, W3D.5, (2015).

[8] I. White et al., *J. Optical Networking*, Vol. **8**, p. 215 (2009).

[9] W. Dally et. al., Morgan Kaufmann, (2004).

[10] S. Liu et. al., *J. Opt. Commun. Netw.*, Vol. **7**, p. A498 (2015).